

**PRIORITY DOCUMENT**  
SUBMITTED OR TRANSMITTED IN  
COMPLIANCE WITH  
RULE 17.1(a) OR (b)

REC'D 24 MAR 2003

WIPO PCT



**Prioritätsbescheinigung über die Einreichung  
einer Patentanmeldung**

**Aktenzeichen:** 102 02 175.9

**Anmeldetag:** 20. Januar 2002

**Anmelder/Inhaber:** PACT Informationstechnologie GmbH, München/DE

**Bezeichnung:** XPP Prozessormodell

**IPC:** G 06 F 15/76

Die angehefteten Stücke sind eine richtige und genaue Wiedergabe der ursprünglichen Unterlagen dieser Patentanmeldung.

München, den 10. März 2003  
Deutsches Patent- und Markenamt  
Der Präsident  
Im Auftrag

Joost

**BEST AVAILABLE COPY**

# XPP Prozessormodell

Volker Baumgarte

19. Januar 2002

## 1 Das klassische Prozessormodell

In dieser Schrift wird ein Prozessormodell für rekonfigurierbare Architekturen vorgeschlagen, das in wesentlichen Punkten an das Modell eines klassischen Prozessors angelehnt ist. Zum besseren Verständnis wird zunächst das klassische Modell näher betrachtet. Dabei wird auf die Betrachtung prozessorexterner Ressourcen (z. B. Hauptspeicher für Programm und Daten etc.) verzichtet.

Ein Prozessor führt in einem *Prozeß* ein *Programm* aus. Das Programm besteht dabei aus einer endlichen Menge von Befehlen (Diese Menge darf Elemente mehrfach enthalten) sowie Informationen über die Reihenfolge, in der Befehle aufeinanderfolgen können. Diese Reihenfolge wird primär über die lineare Anordnung der Befehle im Programmspeicher und die Ziele von Sprungbefehlen festgelegt. Befehle werden dabei üblicherweise über ihre Adresse identifiziert. Als Beispiel zeigt Abbildung 1 (a) ein Programm in VAX-Assembler zur Exponentiation.

Man kann ein Programm auch als gerichteten Graphen auffassen, wobei die Befehle die Knoten bilden, und die Reihenfolge als Kanten des Graphen modelliert wird. Dieser Graph wird in Abbildung 1 (b) gezeigt. Der Graph besitzt dabei einen eindeutigen Start- und einen eindeutigen Endknoten. (Im Bild nicht gezeigt, durch die Pfeile angedeutet.) Die Kanten können zusätzlich mit Übergangswahrscheinlichkeiten markiert werden. Diese Information kann dann zur Sprungvorhersage genutzt werden.

Durch die lineare Anordnung der Befehle im Speicher ergeben sich mehr Abhängigkeiten als unbedingt notwendig. So sind z. B. im gezeigten Beispiel die Befehle DECL und MULL2 voneinander unabhängig. Dies geht aus dem Graphen in Abbildung 1 (b) nicht hervor. Das Modell kann entsprechend erweitert werden durch Aufteil- und Zusammenfassungsknoten. Dies ist in Abbildung 1 (c) gezeigt. Heutige Prozessoren erkennen derartige Möglichkeiten der Parallelausführung zum Teil in Hardware und verteilen die Operationen auf verschiedene Rechenwerke.

Für die weiteren Betrachtungen wird das Modell aus Abbildung 1 (b) verwendet. Die Behandlung der zusätzlichen Komplexität des Aufteilens und Zusammenfassens wird auf einen späteren Zeitpunkt verschoben.

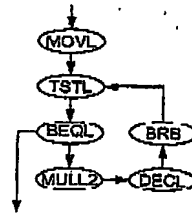
Ein Prozeß braucht zu seiner Ausführung außer dem Programm weitere Ressourcen. Diese sind innerhalb des Prozessors die Register und die Status-Flags.

```

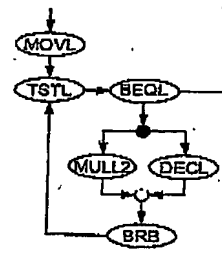
    MOVL #1, R2
10$: TSTL R1
    BEQL 20$
    MULL2 R2, R0
    DECL R1
    BRB 10$
20$:

```

(a) VAX-Assembler



(b) direkter Graph



(c) paralleler Graph

Abbildung 1: Einfaches Beispielprogramm (Exponentiation)

Der Program Counter (PC) ist nicht Teil dieser Ressourcen, er wird lediglich benötigt, um eine Teilmenge der Kanten des Graphen zu realisieren.

Die oben angesprochenen Ressourcen dienen dazu, Informationen zwischen den einzelnen Befehlen des Programms zu übermitteln.

Es ist Aufgabe des Betriebssystems, dafür zu sorgen, daß einem Prozeß die zu seiner Ausführung benötigten Ressourcen zur Verfügung stehen und bei seiner Beendigung wieder freigegeben werden. Heutige Prozessoren besitzen üblicherweise nur einen Registersatz, so daß nur ein Prozeß gleichzeitig auf dem Prozessor ablaufen kann.

Man kann sich leicht vorstellen, daß die Befehle von zwei unterschiedlichen Prozessen in beliebiger Reihenfolge durchmischt ausgeführt werden können, solange beide Prozesse disjunkte Ressourcen verwenden (z. B. Prozeß 1 verwendet die Register 0-3, Prozeß 2 Register 4-7).

Befehle eines Prozessors haben üblicherweise die folgenden Eigenschaften:

- Ein Befehl wird während der Ausführung nicht unterbrochen.
- Die Ausführungszeit aller Befehle überschreitet einen gewissen Maximalwert nicht.
- Ungültige Befehle werden vom Prozessor erkannt.

## 2 Übertragung des Modells auf die XPP-Architektur

Die XPP-Architektur ist eine rekonfigurierbare Prozessorarchitektur, die wesentlich durch die Patent(e)(anmeldungen) PACT01, 02, 03, 04, 05, 07, 08, 09, 10, 13, 17, 22, 23, 24 definiert ist. Diese Schriften werden zu Offenbarungszwecken vollumfänglich eingeleistet. Ebenfalls wird auf PACT11, 20, 27 verwiesen, in denen entsprechende Hochsprachen-Compiler beschrieben sind, sowie auf PACT 21, worin ein entsprechender Debugger beschrieben ist. Auch diese Schriften werden

zu Offenbarungszwecken vollständig eingegliedert. Sämtliche offenen Prioritäten aus den vorgenannten Schriften werden hiermit beansprucht.

Der klassische Befehl wird ersetzt durch eine Konfiguration im bekannten Sinne, im folgenden Komplex-Befehl (Complex Instruction Word, CIW) genannt. Die Kanten des Graphen in Abbildung 1 (b) werden realisiert durch Events an die CM-Ports. Damit kann ein vollständiges Programm realisiert werden.

Für den Registersatz eines klassischen Prozessors gibt es zur Zeit noch keine Entsprechung auf der XPP-Architektur. Die benötigten Eigenschaften lassen sich aber dennoch beschreiben:

- Da die XPU im wesentlichen auf Datenströmen arbeitet, muß ein Register in der Lage sein, einen Datenstrom bzw. Teile davon zu speichern.
- Ein Register muß alloziert und freigegeben werden können. Dabei muß es so lange belegt bleiben wie das Programm auf der XPU läuft. (HW-Unterstützung der Ressourcenverwaltung des Betriebssystems.)
- Gleichzeitiges Lesen und Schreiben (Read-Modify-Write) desselben Registers sollte möglich sein.

Es wird vorgeschlagen, entsprechend modifizierte RAM-Objekte zu verwenden. Diese sollen zunächst als Register verwendet werden. Eine ausführliche Beschreibung des Register-Objekts findet sich in Abschnitt 4.

Eine Konfiguration (CIW) wird in dem Moment vom Array entfernt, in dem sie über einen Event an einen CM-Port das nächste CIW anfordert. Der Reconfig-Event kann dabei entweder über den Reconfig-Port eines ALU-Objekts oder implizit vom Port-Objekt erzeugt werden. In späteren Versionen sollte dies grundsätzlich vom Port-Objekt aus erfolgen.

Ebenso wie Befehle auf einem klassischen Prozessor nicht unterbrochen werden, läuft auch ein CIW auf der XPU ohne Unterbrechung bis es das nächste CIW über einen CM-Port anfordert. Es wird nicht vorzeitig beendet. Um dennoch einen regelmäßigen Befehlswechsel sicherstellen zu können (dieser wird später für Multitasking benötigt), wird die maximale Ausführungszeit eines CIW nach oben beschränkt. Damit wird die zweite Eigenschaft eines Befehls gefordert. Es ist Aufgabe des Compilers, dafür zu sorgen, daß jedes erzeugte CIW dieser Bedingung genügt. Ein CIW, das diese Bedingung verletzt, ist ein ungültiger Befehl. Er kann von der Hardware während der Ausführung z. B. über einen Watchdog-Timer erkannt werden. In diesem Fall wird das ungültige CIW über einen Reconfig beendet und eine Exception an das Betriebssystem geschickt.

Da die CIWs sehr lang sind, sind dementsprechend auch die Instruction-Fetch- (Config-Request bis Code ist im FILMO-Cache) und Instruction-Decode-Zeiten (Verteilung der Config-Daten vom FILMO-Cache in die Config-Register der Objekte) sehr lang. Dadurch ist die Auslastung der Execution Units (Array) durch einen Prozeß nicht sehr hoch. Wie dieses Problem mit mehreren Prozessen gelöst werden kann, wird in Abschnitt 6 gezeigt.

3

\* Sämtliche rekonfigurierbaren Bausteine, i.B. die der XPP-Architektur, werden nachfolgend zusammenfassend mit XPU bezeichnet.

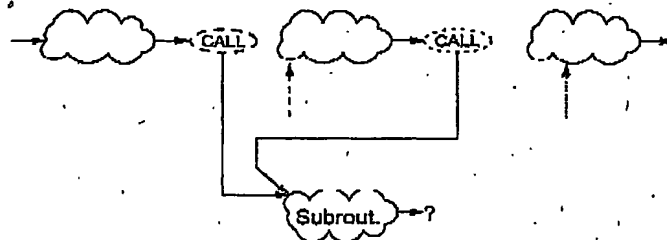


Abbildung 2: Beispiel für ein Unterprogramm mit Aufruf

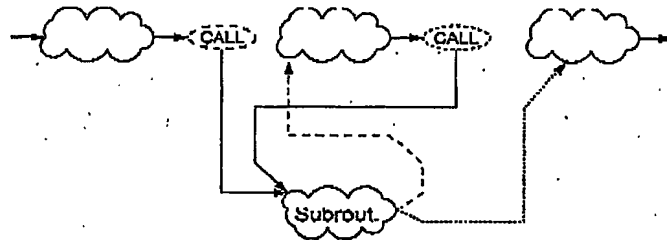


Abbildung 3: Dynamisch veränderter Graph während der Ausführung

### 3 Unterprogramme

Ein Unterprogramm in der Graphendarstellung ist ein Teilgraph eines Programms mit eindeutig bestimmten Eingangsknoten. Die Kante des Unterprogrammaufrufs innerhalb des Graphen ist dadurch statisch bekannt. Die weiterführende Kante am Ausgangsknoten des Unterprogramms ist jedoch nicht statisch bekannt. In Abbildung 2 wird dies verdeutlicht. Die Kanten vom Hauptprogramm zum Unterprogramm sind vorhanden, die Fortführung nach dem Unterprogramm ist jedoch im Unterprogramm nicht bekannt. Die jeweilige Fortführung ist fest mit dem Unterprogrammaufruf verbunden (durch gestrichelte bzw. gepunktete Linien markiert). Sie muß vor dem Erreichen des Eingangsknotens in geeigneter Weise in den Graphen eingefügt werden. Dies ist in Abbildung 3 verdeutlicht.

In klassischen Prozessoren geschieht dies üblicherweise dadurch, daß beim Unterprogrammaufruf (Call) die Adresse des auf das Unterprogramm folgenden Befehls (das ist genau die fehlende Kante) auf einem Call-Stack abgelegt wird. Von dort kann sie beim Rücksprung (Return) geholt werden.

Übertragen auf die XPU wird also ein Stack-Objekt benötigt. Dies ist genauso wie die Register-Objekte eine Prozeß-Ressource und wird genauso verwaltet. Das CIW, das bei seiner Beendigung den Unterprogrammaufruf veranlaßt, konfiguriert die Rücksprung-Kante auf das Stack-Objekt. Durch einen Event veranlaßt das

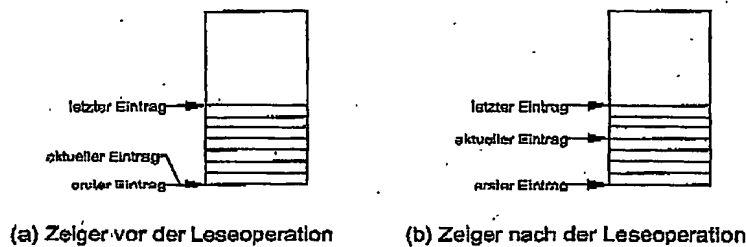


Abbildung 4: Beispiel für eine Register-Leseoperation

letzte CIW des Unterprogramms das Stack-Objekt, den obersten Eintrag vom Stack zu entfernen und als Config-Request an den CM zu schicken.

Eine Implementierung kann beispielsweise zunächst über eine Software-Lösung innerhalb des CM erfolgen. Dabei wird eine spezielle Config-ID (z. B. -1) als Rücksprung reserviert. Wenn der CM diese ID erhält, ersetzt er sie durch den obersten Eintrag seines lokal verwalteten Stacks. Später kann dies durch ein eigenes Objekt oder ein entsprechend erweitertes Port-Objekt erfolgen.

Stack-Overflow sowie Stack-Underflow sind Exceptions, die an das Betriebssystem weitergegeben werden.

#### 4 Das Register-Objekt

Ein klassisches Prozessorregister enthält zu jedem Zeitpunkt ein Datenwort. Ein Befehl kann den Registerinhalt lesen, schreiben oder verändern (Read-Modify-Write). Ein XPU-Register sollte die gleichen Eigenschaften aufweisen, allerdings enthält es statt eines einzelnen Wertes einen Wertevektor oder Teile davon.

Üblicherweise dürfte die Organisation eines XPU-Registers als eine Art FIFO die günstigste Möglichkeit sein. In bestimmten Fällen kann aber auch wahlfreier Zugriff erforderlich werden. Im folgenden werden die drei oben angesprochenen Registerzugriffe im einzelnen erläutert. Dabei wird ein wahlfreier Zugriff nicht betrachtet.

**Lesezugriff:** Beim Start eines CIW enthält das Register einen Datenvektor unbekannter Länge. Die einzelnen Elemente des Vektors werden sequentiell entnommen. Dabei wird beim letzten Element des Vektors ein Event generiert, der anzeigt, daß das Register jetzt leer ist und das CIW terminieren kann.

Der Zustand des Registers kann dabei mit 3 Zeigern charakterisiert werden, die zeigen auf den ersten, letzten und aktuellen Eintrag im Datenvektor. Die Stellung der Zeiger zu Beginn eines CIW wird beispielhaft in Abbildung 4 (a) gezeigt. Dabei steht der Zeiger für den aktuellen Eintrag auf dem ersten Eintrag.

In Abbildung 4 (b) zeigt in einem Beispiel, wie die Zeigerstellung eines Registers am Ende eines CIW aussehen kann. Im dort gezeigten Fall wurde der Vektor nicht vollständig gelesen.

Nun muß entschieden werden, was mit dem Registerinhalt geschieht. Es gibt die folgenden Möglichkeiten:

- Das Register wird geleert. Alle nicht verarbeiteten Daten werden gelöscht. Der Zeiger für den aktuellen Eintrag wird auf den letzten Eintrag gesetzt.
- Das Register wird auf den Ursprungszustand zurückgesetzt. Dadurch kann das nächste CIW wieder auf den vollen Datenvektor zugreifen. Der Zeiger für den aktuellen Eintrag wird auf den ersten Eintrag zurückgesetzt.
- Nur die bereits gelesenen Daten werden aus dem Register entfernt. Die ungelesenen Daten stehen für das nächste CIW zur Verfügung. Die Zeiger werden dabei nicht verändert.

Im Anschluß daran werden die Werte zwischen dem ersten Eintrag und dem aktuellen Eintrag aus dem Register entfernt. Sie stehen für weitere Operationen nicht mehr zur Verfügung.

Die dritte Möglichkeit ist insbesondere dann interessant, wenn ein CIW aufgrund der maximalen Ausführungszeit für ein CIW den Datenvektor nicht vollständig verarbeiten kann. Siehe hierzu auch Abschnitt 7.

**Schreibzugriff** Hier werden Daten sequentiell in das Register geschrieben. Dabei wird ein Event generiert, wenn der Füllstand des Registers einen bestimmten Wert erreicht. Je nach CIW kann dies eine der folgenden Möglichkeiten sein:

- Das Register ist vollständig gefüllt.
- Es sind noch genau  $n$  Einträge im Vektor frei. Dies berücksichtigt die Latenzzeit im CIW, durch die noch  $n$  Werte nach dem Event auf das Register laufen.
- Das Register ist zu  $m\%$  gefüllt.

Ein CIW, das versucht, auf ein vollständig gefülltes Register zu schreiben, ist ungültig und wird mit einer Exception beendet (Illegal Opcode).

Beim Start des CIW muß festgelegt sein, in welchem Zustand sich das Register befindet. Abbildung 5 (a) zeigt ein Register vor einem Schreibzugriff, das noch Daten enthält. Bestehende Daten können gelöscht werden, so daß der Schreibzugriff mit einem leeren Vektor beginnt (Abbildung 5 (b)). Alternativ können die geschriebenen Daten auch an den bestehenden Inhalt angehängt werden. Dies zeigt Abbildung 5 (c). Dies ist dann interessant, wenn das vorhergehende CIW nicht den kompletten Vektor erzeugen konnte aufgrund der maximalen Ausführungszeit.

Der Zustand des Registers nach erfolgter Schreiboperation zeigt Abbildung 5 (d) bzw. (e). Die neu geschriebenen Daten sind dabei schraffiert gekennzeichnet.

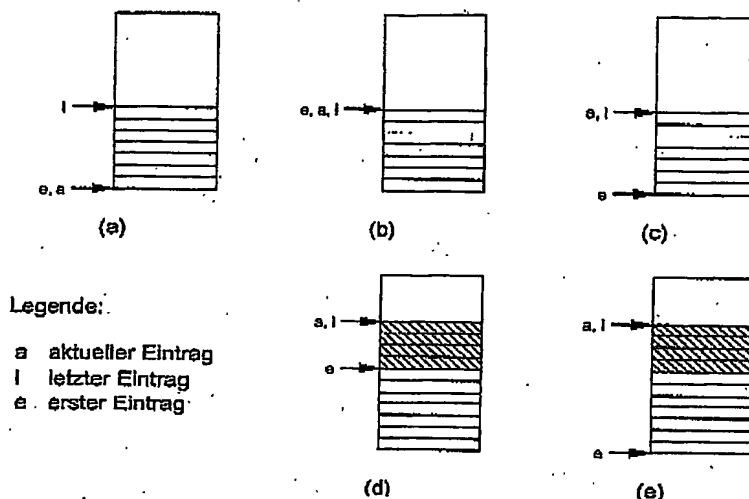


Abbildung 5: Beispiel für eine Register-Schreiboperation

**paralleler Schreib-/Lesezugriff** Die Beschränkung auf reine Schreib- bzw. Lesezugriffe erfordert eine höhere Registerzahl als nötig. Wenn einem Register durch Lesezugriffe Daten entnommen werden, entsteht dadurch Platz, in dem Schreibdaten aufgenommen werden können. Es muß lediglich sichergestellt werden, daß geschriebene Daten nicht vom gleichen CIW wieder gelesen werden können. Dazu wird in den FIFO eine virtuelle Trennlinie eingeführt. Das Register wurde vollständig gelesen, wenn diese Trennlinie am Ausgang des FIFOs angelangt ist. Kann ein Schreibzugriff für ein Datenwort nicht ausgeführt werden, weil das Register noch mit ungelesenen Lesedaten blockiert ist, wird das CIW beendet und eine Illegal Opcode Exception erzeugt. Das Verhalten des Registers ist ansonsten genauso wie bei Schreib- und Lesezugriff erläutert.

Zusätzlich muß spezifiziert werden, was mit der virtuellen Trennlinie zwischen Lese- und Schreibdaten geschieht. Diese kann entweder an der Stelle verbleiben, wo sie gerade steht. Dies ist dann nützlich, wenn ein CIW aufgrund der Zeitbeschränkung beendet werden muß. Alternativ wird die Trennlinie an das Ende aller Daten gesetzt.

Kombinierte Schreib-/Lesezugriffe sind problematisch, wenn das CIW mit einer Exception beendet wurde. In diesem Fall ist es nicht mehr möglich, die Register auf ihre Werte beim Start des CIW zurückzustellen. Das Debugging wird dadurch mindestens erschwert, siehe auch Abschnitt 8.

Abbildung 6 zeigt die Funktionsweise an einem Beispiel. Zu Beginn enthält das Register Daten (a), die im folgenden teilweise (b) bzw. vollständig (c) gelesen werden. Neu geschriebene und gelesene Einträge sind dabei durch Schraffur



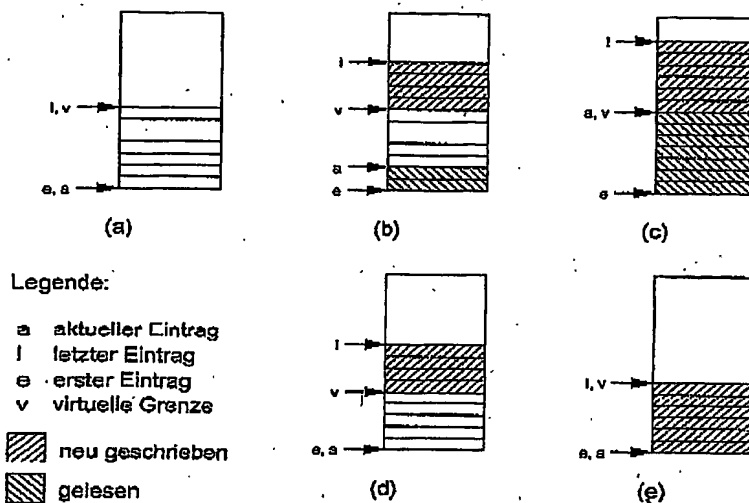


Abbildung 6: Beispiel für eine Register-Schreib-/Leseoperation

gekennzeichnet. Die Teilbilder (d) und (e) zeigen den Zustand des Registers nach dem notwendigen Zeiger-Update. Dies ist kein expliziter Schritt, sondern wird hier nur zur Verdeutlichung dargestellt. Die gelesenen Einträge müssen sofort entfernt werden, um Platz für die neu zu schreibenden Einträge zu schaffen.

Ein Prozeß muß jedes benötigte Register allozieren, bevor er es verwenden kann. Dies geschieht über ein zweites CM-Interface des Registers. Dort wird auch eingetragen, zu welchem Prozeß das Register jetzt gehört. Diese Konfiguration bleibt auch über implizite Reconfigs hinweg erhalten. Das Register muß explizit vom CM freigegeben werden. Dies geschieht bei der Beendigung eines Prozesses.

Mit der Konfiguration jedes CIWs muß den Registern mitgeteilt werden, zu welchem Prozeß das CIW gehört. Dies ermöglicht das Umschalten zwischen mehreren Registersätzen. Das Verfahren wird genauer im Abschnitt 6 beschrieben.

## 5 Interrupts

Bei Interrupts muß zwischen zwei unterschiedlichen Typen unterschieden werden. Zum einen gibt es die Hardware-Interrupts, wo der Prozessor auf ein externes Ereignis reagieren muß. Diese werden üblicherweise vom Betriebssystem bearbeitet und sind für die laufenden Prozesse nicht sichtbar. Sie sollen hier nicht weiter behandelt werden.

Der zweite Typ sind die Software-Interrupts. Diese werden häufig benutzt, um

asynchrone Interaktionen zwischen Prozeß und Betriebssystem zu realisieren. So ist es z. B. unter VMS möglich, eine Leseanforderung an das Betriebssystem zu schicken, ohne auf die eigentlichen Daten zu warten. Sobald die Daten vorhanden sind, unterbricht das Betriebssystem das laufende Programm und ruft asynchron eine Prozedur des Programms auf. Dieses Verfahren nennt sich dort Asynchronous System Trap (AST).

Dieses Verfahren kann in gleicher Form auf der XPU angewendet werden. Hierzu ist Unterstützung im CM vorzusehen. Der CM weiß, ob eine asynchrone Routine für einen Prozeß aufgerufen werden muß. In diesem Fall wird der nächste Request, der vom Array kommt, nicht direkt abgearbeitet, sondern gespeichert. Stattdessen wird eine Folge von CIWs eingefügt, die zunächst die Prozessorstatus (die Registerinhalte) sichern, die asynchrone Routine ausführen und dann die Registerinhalte wiederherstellen. Im Anschluß daran kann der ursprüngliche Request abgearbeitet werden.

## 6 Multitasking

In Abschnitt 2 wurde festgestellt, daß die XPU-Architektur mit nur einem Prozeß nicht ausgelastet werden kann aufgrund der Länge der CIWs sehr hohe Lade- und Dekodierungszeiten auftreten. Dieses Problem kann durch die gleichzeitige Ausführung mehrerer Prozesse gelöst werden. Hierbei werden auf der XPU mehrere Registersätze vorgesehen, damit beim Kontextwechsel einfach zwischen den Registersätzen umgeschaltet werden kann und keine aufwendigen Register-Freiräum- und -Lade-Operationen erforderlich werden.

Während der Ausführung von Befehlen anderer Prozesse steht genügend Zeit zur Verfügung, um den nächsten Befehl des aktuellen Prozesses zu holen und über den FILMO an die Objekte zu verteilen. Die optimale Anzahl an Registersätzen muß dabei in Abhängigkeit von der durchschnittlichen Ausführungszeit eines CIW und den durchschnittlichen Lade- und Dekodierungszeiten der CIWs bestimmt werden. Dabei kann Latenzzeit problemlos durch eine größere Anzahl Registersätze abgefangen werden. Wichtig für die Funktion des Verfahrens ist, daß die durchschnittliche CW-Laufzeit größer ist als die jeweils effektiv benötigte Zeit zum Laden bzw. Dekodieren des CIW.

Die korrespondierenden Register der unterschiedlichen Registersätze liegen dabei für den Programmierer auf derselben Objektadresse. Dabei können zu jedem Zeitpunkt immer nur die Register eines Registersatzes verwendet werden.

Der Kontextwechsel zwischen den Registersätzen kann dadurch realisiert werden, daß vor jedem CIW der entsprechende Kontext an die Objekte übertragen wird. Dieses Verfahren wurde bereits in Abschnitt 4 angesprochen.

Im allgemeinen werden auf einem System mehr Prozesse vorhanden sein als Registersätze auf dem Prozessor. Das bedeutet, daß gelegentlich ein Prozeß vom Prozessor entfernt werden muß. Dazu wird ähnlich wie beim Software-Interrupt

eine Kante den Programmgraphen vom CM aufgetrennt. Die Registerinhalte des Prozesses werden gesichert und die vom Prozeß allozierten Prozessorressourcen (Register, Stack-Objekt, etc.) wieder freigegeben. Die so freigewordenen Ressourcen werden nun von einem anderen Prozeß alloziert. Dann werden die für diesen Prozeß gespeicherten Registerinhalte wieder zurückgeschrieben und der Prozeß an dessen aufgetrennter Kante fortgesetzt.

Das Sichern und Rückladen der Registerinhalte kann dabei über CIWs erfolgen.

## 7 CIW und Schleifen

Aufgrund der oben geforderten Eigenschaft, daß ein CIW spätestens nach einer gewissen Maximalanzahl an Takten terminieren muß, können allgemeine Schleifen nicht ohne weiteres in ein CIW übersetzt werden. Es ist immer möglich, den Schleifenrumpf in ein CIW zu übersetzen und die Schleifenkontrolle über Rekonfiguration abzuwickeln. Dies kostet jedoch jegliche Performance. Dieser Abschnitt zeigt, wie eine Schleife so umgeformt werden kann, daß die Anzahl der Rekonfigurationen minimiert wird.

Im folgenden wird von folgendem Programmstück ausgegangen:

```
while (condition) {
    something;
}
```

Dabei soll sowohl die Laufzeit von condition wie something bestimmt oder nach oben abgeschätzt werden können. Die Schleife kann nun wie folgt umformuliert werden:

```
while (1) {
    if (!condition) goto finish;
    something;
}
finish:
```

Nun kann der Schleifenrumpf so oft iteriert werden wie es die maximale Laufzeit eines CIW zuläßt. Hierzu wird eine neue Variable z eingeführt, die weder in condition noch in something vorkommt. Das Programm sieht nun folgendermaßen aus:

```
while (1) {
    for (z=0; z<MAX; z++) {
        if (!condition) goto finish;
        something;
    }
}
finish:
```

Die for-Schleife besitzt eine vom Compiler bestimmbare maximale Laufzeit. Sie kann deshalb auf ein CIW abgebildet werden. MAX wird vom Compiler in Abhängigkeit von der maximalen Laufzeit und den Einzellaufzeiten der Anweisungen bestimmt.

Das so entstehende CIW hat zwei Ausgangskanten. Der Ausgang über das goto führt zum nächsten CIW, der Ausgang über das reguläre Ende des for bildet eine Kante auf sich selbst. Darüber wird die Endlosschleife realisiert.

## 8 Debugging

Im klassischen Prozessor erfolgt das Debugging auf Befehlsbasis, d. h. der Ablauf eines Programms kann jederzeit zwischen zwei Befehlen unterbrochen werden. An diesen Unterbrechungspunkten hat der Programmierer Zugriff auf die Register. Er kann sie ansehen und modifizieren. Unterbrechungspunkte können auf verschiedene Art und Weise realisiert werden. Zum einen kann das Programm modifiziert werden, d. h. der Befehl, vor dem angehalten werden soll, wird durch andere Befehle ersetzt, die den Debugger aufrufen. Im Graphenmodell entspricht dies dem Austausch eines Knotens durch einen anderen Knoten oder einen Teilgraphen. Eine andere Methode beruht auf zusätzlicher Hardware-Unterstützung. Hierbei wird dem Prozessor mitgeteilt, bei welchem Befehl das Programm unterbrochen werden soll. Der entsprechende Befehl wird dabei üblicherweise über seine Adresse identifiziert.

Beide Möglichkeiten stehen auch auf der XPU zur Verfügung. Ein CIW kann jederzeit vom Debugger durch ein anderes CIW ersetzt werden. Dieses CIW kann z. B. die Registerinhalte in den Hauptspeicher kopieren, wo diese entweder mit einem XPU-externen Debugger analysiert werden können. Alternativ kann der Debugger auch auf der XPU ablaufen. Weiterhin kann eine Hardware-Unterstützung im CM vorgesehen werden, die CIWs bei deren Request anhand der ID identifiziert und dann den Debugger aufruft.

Zusätzlich kann eine Unterbrechung auch an einer Kante des Graphen festgemacht werden, da diese im Gegensatz zu klassischem Programmcode explizit vorliegen.

Die oben beschriebene Art des Debugging ist für klassische Prozessoren vollständig ausreichend, da die Befehle zumeist sehr einfach sind. Eine hinreichend feine Auflösung der beobachtbaren Punkte ist gegeben. Weiterhin kann sich der Programmierer darauf verlassen, daß die einzelnen Befehle korrekt sind. (Dafür sorgt üblicherweise der Prozessorhersteller.) Auf der XPU hingegen definiert der Programmierer sich die Prozessorbefehle selbst. Dementsprechend können die so definierten Befehle fehlerhaft sein. Ein Debugging der einzelnen Befehle wird also erforderlich, im folgenden als Microcode-Debugging bezeichnet.

Microcode-Debugging erfordert einen erheblichen Aufwand, da der Programmierer Zugriff auf alle internen Register und Datenpfade des Prozessors haben muß. Eine Hardware-Unterstützung hierfür ist sehr aufwendig und zu reinen De-

bugging-Zwecken nicht sinnvoll. Alternativ kann der Zustand des Prozessors vor dem fraglichen Befehl gesichert werden und die Ausführung des eigentlichen Befehls auf einem Software-Simulator erfolgen.

Bei geeignetem Programmiermodell kann der Debugger auch bei Auftreten einer Exception innerhalb eines Befehls aufgerufen werden. Hierzu ist es notwendig, daß die Register auf den Zustand vor dem Start des Befehls zurückgestellt werden können und sonst keine Seiteneffekte aufgetreten sind. Dann kann der fragliche Befehl im Software-Simulator gestartet werden und bis zum Auftreten der Exception simuliert werden.

## 9 Verteilter Konfigurations-Cache

Aufgrund des zentralen Konfigurations-Caches beim FILMO dauert es verhältnismäßig lange, bis eine Konfiguration auf die einzelnen Objekte einer PAC verteilt ist. Dieser Abschnitt beschreibt nun ein Verfahren, um diese Zeit abzukürzen.

Dazu erhält jedes Objekt seinen eigenen lokalen Cache. Dieser speichert die Konfigurationsdaten verschiedener Konfigurationen für genau dieses Objekt. Auch die Tatsache, daß ein Objekt von einer Konfiguration keine Daten erhalten hat, wird gespeichert. Für jede angeforderte Konfiguration kann der Cache dadurch eine der folgenden Aussagen treffen:

- Die Konfigurationsdaten sind im Cache vorhanden.
- Für diese Konfiguration werden keine Daten benötigt.
- Über diese Konfiguration ist nichts bekannt.
- Es werden Konfigurationsdaten benötigt, diese sind jedoch nicht im Cache verfügbar. (z. B. aufgrund der Länge der Konfiguration, RAM-Preload etc.)

Die letzten beiden Aussagen können dabei zusammengefaßt werden. Bei beiden muß der Code oder die Tatsache, daß kein Code benötigt wird angefordert werden.

Ein Auftrag für eine Konfiguration wird dabei vom FILMO als Broadcast auf dem Testbus an alle Objekte verschickt. Wenn alle Objekte die Konfiguration in ihrem lokalen Cache haben, kann diese per Broadcast auf dem Configbus gestartet werden. Im Idealfall erfordert der Start einer Konfiguration also nur die Übertragung eines einzigen Konfigurationswortes.

Sollte ein Objekt nicht über die Konfigurationsdaten verfügen, muß dieses an dem FILMO zurückgemeldet werden. Im einfachsten Fall geschieht dies über ein Reject auf der vorhandenen Leitung. Der FILMO weiß dann aufgrund dieses Signals, daß mindestens einem Objekt der PAC die Konfigurationsdaten fehlen. Er kann dann die kompletten Daten übertragen. Alternativ kann auch jedes Objekt getrennt einen Request für die Daten auslösen. Hier muß ein geeigneter Kompromiß zwischen der Anzahl der Requests und der Menge der zu übertragenden Konfigurationsdaten gefunden werden. Kleine PAC-Größen sind dabei auch aufgrund der geringeren Latenz auf dem Konfigurationsbus von Vorteil.

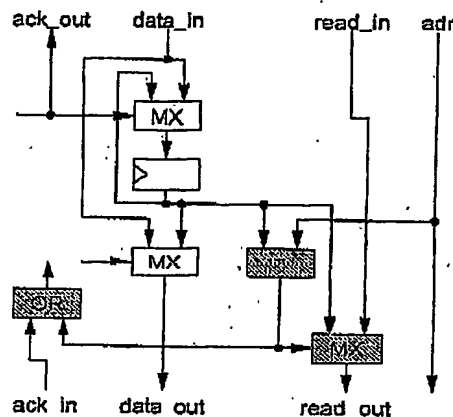


Abbildung 7: modifizierte FIFO-Stufe

**Aufbau des Caches** Ein Cache besteht immer aus zwei Teilen. Der eine Teil enthält die eigentlichen Daten (hier die Konfigurationsworte), der andere Teil enthält Verwaltungsinformation (hier die enthaltenen Konfigurationsnummern sowie ihr Alter). Zunächst wird der Verwaltungsteil beschrieben.

Wünschenswert ist, daß die am längsten nicht mehr benutzte Konfiguration aus dem Cache entfernt wird, wenn dies notwendig wird. Solange nur neue Konfigurationen angefordert werden, sind die Einträge im FIFO korrekt sortiert. Sollte eine Konfiguration angefordert werden, für die bereits ein Eintrag im FIFO vorhanden ist, muß dieser Eintrag aus dem FIFO entfernt werden. Er wird dann wieder neu am Ende eingefügt. Eine für diesen Zweck modifizierte FIFO-Stufe zeigt Abbildung 7. Die schraffierten Module sind dabei zusätzlich zu einer normalen FIFO-Stufe. Sie vergleichen den Inhalt der Stufe mit der geforderten Konfigurationsnummer und erzeugen bei Gleichheit ein ack für die Stufe. Damit werden die Daten der Stufe gelesen und alle anderen Werte rutschen um eine Stufe nach.

Die Einträge in diesem FIFO enthalten außer der Konfigurationsnummer weitere Informationen. Dies ist entweder ein Zeiger (Adresse) auf die Konfigurationsdaten oder eine der beiden Möglichkeiten „keine Daten erforderlich“ (z.B. als 0 codiert) bzw. „Daten müssen angefordert werden“ (z.B. -1). Die Zusammenschaltung mehrerer Stufen zeigt Abbildung 8.

Dabei wird die Read-Chain mit der geforderten Konfigurationsnummer und dem Status -1 initialisiert. Dieser Wert kommt unverändert genau dann am Ausgang der Read-Chain heraus, wenn die Konfigurationsnummer nicht im FIFO gespeichert ist. Damit kann der Ausgang der Read-Chain in jedem Fall dazu verwendet werden, die Konfigurationsnummer auf den FIFO zu schreiben. Das Signal `ack_in` wird dann aktiviert, wenn der FIFO voll ist und sich die gesuchte Konfigurations-

*beispielhaft die*

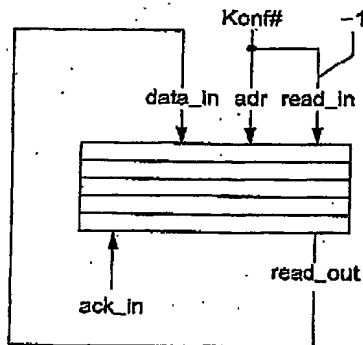


Abbildung 8: Verwaltungs-FIFO für lokalen Cache

nummer nicht im FIFO befindet. Dies ist der einzige Fall, wo der älteste Eintrag aus dem FIFO entfernt werden muß, weil der Verwaltungsspeicher voll ist.

Der eigentliche Datenspeicher wird aufgrund der unterschiedlichen Anzahl der Konfigurationswörter pro Konfiguration als verkettete Liste organisiert. Andere Implementierungen sind denkbar. Eine verkettete Liste kann als RAM einfach dadurch implementiert werden, daß zusätzlich zu den Daten jeweils die Adresse des nachfolgenden Datenwortes abgelegt wird. Zusätzlich zu den Listen für die eigentlichen Konfigurationen wird eine Frei-Liste geführt, in der alle nicht benutzten Einträge aufgeführt sind. Diese muß nach einem Reset zunächst initialisiert werden.

Abbildung 9 zeigt einen möglichen Cache-Inhalt während des Betriebs. Freie Einträge im Datenspeicher sind dabei weiß, von einer Konfiguration belegte schraffiert unterlegt. Konfigurationen müssen dabei nicht auf aufeinanderfolgenden Adressen liegen. Konfiguration 18 besitzt keine Konfigurationsdaten, deshalb führt auch kein Zeiger in den Datenspeicher.

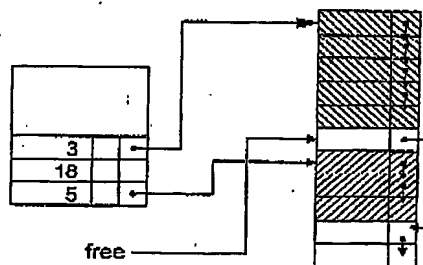


Abbildung 9: Beispiel für einen Cache-Zustand

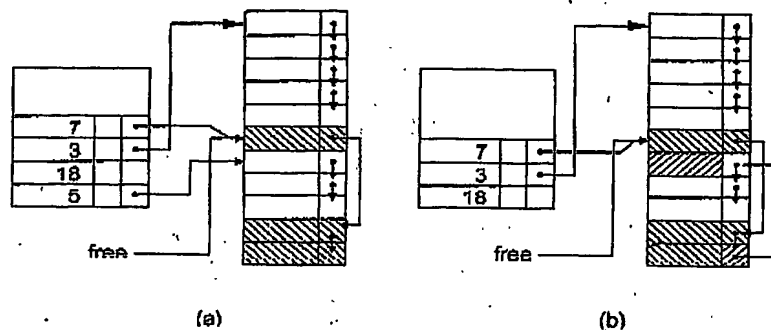


Abbildung 10: Beispiel für die Entfernung eines Eintrages

Eine neue Konfiguration wird auf die Freiliste in den Datenspeicher geschrieben. Dabei wird die Zeigerinformation des Datenspeichers nicht modifiziert. Lediglich für das letzte Datenwort einer Konfiguration wird die Zeigerinformation verändert, um anzuzeigen, daß die Liste jetzt hier ändert. Der Zeiger auf die Freiliste wird auf den nächsten Eintrag gesetzt.

Es kann passieren, daß der Platz in der Freiliste nicht ausreicht, um die ankommenden Konfigurationsdaten vollständig aufzunehmen. In diesem Fall muß entschieden werden, ob eine alte Konfiguration aus dem Datenspeicher entfernt wird oder ob die aktuelle Konfiguration nicht in den Cache aufgenommen werden soll. Im letzteren Fall werden die nachfolgenden Konfigurationswörter verworfen. Da keine Zeiger verändert wurden, bleibt die Freiliste wie zuvor, lediglich einige unbenutzte Datenwörter besitzen einen anderen Wert.

Die Entscheidung, welche Konfiguration nicht mehr im Cache liegen soll (die älteste oder die aktuelle) kann an der Anzahl der bereits geschriebenen Konfigurationswörter festgemacht werden. Es macht wenig Sinn, mehrere gecachte Konfigurationen zu entfernen, um Platz für z. B. eine lange RAM-Initialisierung zu schaffen.

Wenn die älteste Konfiguration entfernt werden soll, wird sie dem FIFO entnommen. Der Zeiger des letzten Eintrags der Freiliste wird auf den dem FIFO entnommenen Wert gesetzt. Ab dieser Adresse kann nun in der gewohnten Weise weiterkonfiguriert werden.

Abbildung 10 zeigt dies an einem Beispiel. Neu konfiguriert werden soll die Konfiguration mit der Nummer 7. In Unterabbildung (a) ist die Freiliste vollständig belegt worden. Es wird entschieden, die älteste Konfiguration (Nr. 5) vom Cache zu entfernen und Konfiguration Nr. 7 weiter auf den Cache zu schreiben. Dazu wird der Zeiger am Ende der Freiliste auf den Start der ehemaligen Konfiguration 5 umgesetzt. Dadurch wird die Freiliste wieder verlängert und es steht wieder Platz für neue Konfigurationswörter zur Verfügung. Die in diesem Schritt betroffenen Speicherteile sind in Unterabbildung (b) gegenidiagonal schraffiert. Bei geeigneter



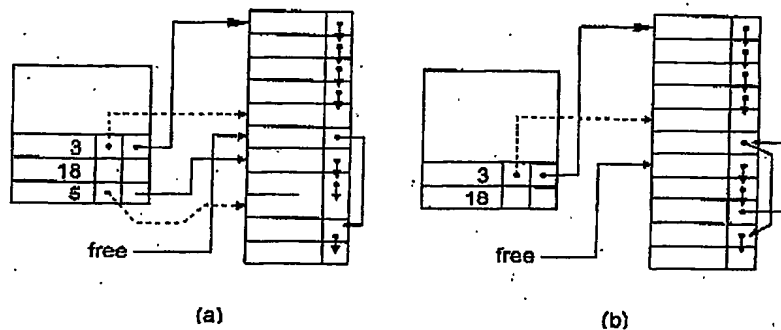


Abbildung 11: Beispiel für das Löschen eines Eintrags

Aufteilung des Speichers kann dies in einem Takt erfolgen.

Mit dem letzten Konfigurationswort wird der entsprechende Zeiger auf Ende gesetzt und der Freizeiger auf den nächsten Eintrag.

Platz im Datenspeicher wird nicht nur dann wieder freigegeben, wenn dieser durch die Aufnahme einer neuen Konfiguration benötigt wird. Auch wenn der Verwaltungsspeicher voll ist und deswegen ein Eintrag vom Verwaltungsspeicher entfernt wird, muß die Freiliste im Datenspeicher angepaßt werden. Hierzu muß entweder der Zeiger am Ende der Freiliste oder am Ende der freiwerdenden Konfiguration angepaßt werden. Beide Informationen stehen bisher nicht zur Verfügung. Man kann sich jetzt durch eine der Listen durchbewegen, bis das Ende erreicht wird. Dies ist jedoch zeitaufwendig.

Alternativ wird im Verwaltungsspeicher ein zusätzlicher Zeiger auf das jeweilige Ende einer Konfiguration abgelegt. Nun ist die Modifikation einfach möglich. Der Freizeiger erhält die Startadresse der alten Konfiguration, und der Zeiger beim letzten Konfigurationswort im Datenspeicher wird auf den Freizeiger gesetzt. Dies wird in Abbildung 11 verdeutlicht. Die Zeiger auf die Konfigurationsenden sind dabei gestrichelt eingezeichnet. Unterabbildung (a) zeigt die Situation vor dem Löschen, Unterabbildung (b) die danach.

+ Verwendung von Fall Through FITA (s. PACTAS), 'ship' und 'empty' - Befehle!

## 10 Optimierung der Busallokation

Derzeit werden die Busse explizit vom Router festgelegt. Das kann dazu führen, daß zwei Konfigurationen sich auf einem Bus überlappen und deswegen nicht gleichzeitig ablaufen können, obwohl insgesamt genügend Busse zur Verfügung stünden.

Algorithmisch macht es keinen Unterschied, über welchen Bus eine Verbindung geführt wird. Deshalb kann man die Busallokation dynamisch von der Hardware durchführen lassen. Eine Konfiguration spezifiziert dazu nur noch, daß sie

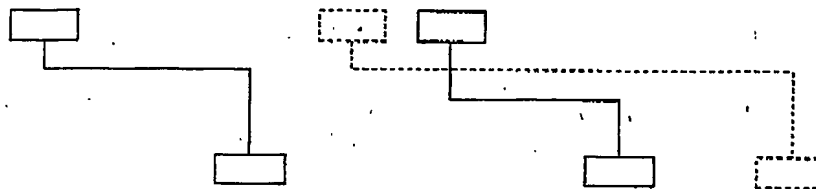


Abbildung 12: Beispiel für eine Busbelegung

eine Verbindung von Punkt A nach Punkt B innerhalb einer Zeile benötigt. Welcher der vorhandenen Busse dann tatsächlich verwendet wird, wählt ein Arbitrier in der Hardware aus. Dieser kann pro Zeile entweder verteilt über Nachbarschaftsbeziehungen oder an zentraler Stelle für die Zeile arbeiten.

Zusätzlich können Busse dynamisch umgelegt werden. Dabei können zwei kurze nichtüberlappende Busse, die aufgrund früherer Belegung auf unterschiedliche Busnummern konfiguriert wurden, bei Freiwerden von Ressourcen auf dieselbe Busnummer umgelegt werden. Dadurch wird Platz geschaffen für zukünftige längere Verbindungen. Dies zeigt Abbildung 12. Hier blockiert die gestrichelte Verbindung den oberen Bus. Nachdem diese freigegeben wurde, blockieren die kurzen Verbindungen immer noch 2 Busse für längere Verbindungen. Durch ein dynamisches Verlegen des unteren Busses um einen Bus nach oben wird dieser wieder vollständig frei.

## 11 Mehrere Objekte auf einer Adresse

Beim Register-Objekt wurde bereits gezeigt, daß es möglich ist, mehrere identische Objekte auf derselben Adresse zu verwalten. Dies ist auch bei den anderen Objekttypen möglich. Algorithmisch ist es z. B. egal, welches FREG bzw. BREG zum Routen verwendet wird. Diese Entscheidung kann also ebenso von der Hardware getroffen werden. Dadurch wird es möglich, die vorhandenen Ressourcen besser auf verschiedene Konfigurationen aufzuteilen, so daß diese dann parallel und gleichzeitig ausgeführt werden können. Ebenso ist es denkbar, z. B. die ALU in mehrfacher Ausführung vorzusehen. Jede Konfiguration darf die ALU nur einmal verwenden. Dadurch wird eine echt gleichzeitige Parallelausführung verschiedener Konfigurationen möglich.

Die mehrfachen Objekte erfordern dabei nicht notwendigerweise zusätzlichen Hardware-Aufwand. Bei geeigneter Wahl von Rechenzeit und Übertragungszeit auf den Bussen kann das Objekt im Zeitmultiplexverfahren arbeiten.

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☒ FADED TEXT OR DRAWING
- ☒ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**